

# Understand modules in python to increase your code's productivity and readability

## Overview

In python, we have modules that are files with a .py extension. It contains functions/classes/variables that we may want to use again and again. So instead of redefining them for the umpteenth time in our program files, we use modules.

## Scope of article

1. The article defines modules and how to use them
2. We will learn different ways to import a module
3. We will also learn some important built-in modules

## Table of content:

1. What are modules in python
2. Types -
  - a. Built-in
  - b. User-defined
3. Create a module
4. How to use a module
5. Variables in module
6. The import statement
7. How to import modules
8. Python module search path
9. Naming a module
10. Re-naming a module

11. Reloading a module
12. The `dir()` built-in function
13. Examples

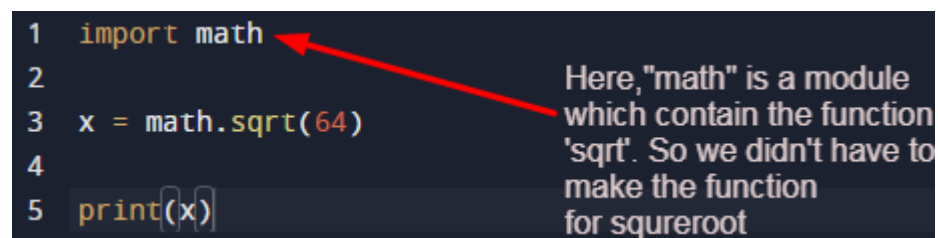
## What are modules in python?

**TLDR: Modules are files with extensions .py. They can contain functions, class, variables, etc.**

Similar to a modular kitchen that is divided into small compartments, an efficient program in python consists of modules. **Modules are handleable units of code written in a separate python file.** This code can include functions, class, variable, etc.

After the module performs its task, the control return back to the calling statement.

They are also called libraries. Modules make the programs less complex, reduce redundancy, and increase readability.



```
1 import math
2
3 x = math.sqrt(64)
4
5 print(x)
```

Here, "math" is a module which contain the function 'sqrt'. So we didn't have to make the function for squareroot

## Types of modules in python

**TLDR: Types of modules in python are built-in and user-defined. Some Important built-in modules are math.py, cmath.py, random.py, and datetime.py.**

In python modules are divided into two types:

1. Built-in modules
2. User-defined modules

Built-in modules: Modules that are pre-available in python are known as built-in modules. They are also known as standard library modules. Here is the list of some built-in modules of python. They need to be imported before use. To import them, write keyword **import** followed by module name.

```
>>> help('modules')

Please wait a moment while I gather a list of all available modules...

__future__      _warnings      idlelib         runfiles
__abc__         _weakref       imaplib         runpy
__ast__         _weakrefset    imghdr         sample
__asyncio       _winapi        imp             sched
__bisect        abc            importlib       secrets
__blake2        aifc           inspect         select
__bootlocale    antigra        interpreterInfo selectors
__bz2           argparse       io             setup_cython
__codecs        array          ipaddress       setuptools
__codecs_cn     ast            itertools       shelve
__codecs_hk     asynchat      json            shlex
__codecs_iso2022 asyncio        keyword          shutil
__codecs_jp     asyncore      lib2to3         signal
__codecs_kr     atexit        linecache       site
__codecs_tw     audioop       locale          six
__collections   base64         logging         smtpd
__collections_abc bdb            lzma            smtplib
__compat_pickle binascii       macpath          sndhdr
__compression   binhex         mailbox          socket
__contextvars   bisect         mailcap          socketserver
__csv           bs4            main             soupsieve
__ctypes        builtins       marshal          sqlite3
```

Here are some important built-in modules in python with their definitions



User-defined modules: These modules are created by the users. User should name their modules while adhering to the following suggestions:

1. It should be short and, lowercased.
2. Avoid using special symbols like( \_,?,, etc). If you name your module “my.module.py” the python expects to find the module.py file in the folder named “my” which is not the case.

## Create a module

To create a module in python write the code in a file with a .py extension.

Example: Here we have created a module named “calc.py” which returns the multiplication and division of two numbers.

```
1 # A simple module, calc.py
2
3 def multiply(x, y):
4     return (x*y)
5
6 def divide(x, y):
7     return (x/y)
8
```

## How to use a module

To use the module file use the keyword “import” followed by the module name

```
import calc.py

return multiply(2,3)
```

## Variables in module

Variables in python tell the computer about the memory location.

Module in python can consist of functions and variables. These variables can be of type arrays, dictionary, objects, etc.

```

1 x=3
2 #Here we used x to define the memory
  location of 3. As we cannot always write
  a fat number for calling x.

```

In python, we don't need to define the datatype of variable used

To declare a variable simply put your variable name which will store the value in L.H.S and enter the value that you want to store in R.H.S.

```

1 # declaring the var
2 Number = 50
3
4 # display
5 print( Number)
6

```

## Rules for naming a variable in python:

1. Variable name must start with a letter or underscore.
2. A variable name cannot begin with a number
3. It can only contain alphanumeric characters or an underscore
4. VAR, var, Var are three different variables as variables in python are case sensitive
5. Keywords reserved in python cannot be used to name the variable.

## How to use variables in modules

### Example

Save this code in the file as mymodule.py

```

1 cat={
2     "name":"cherry",
3     "age=2",
4     "house":"streets"
5 }

```

Import the module mymodule.py and access the cat dictionary:

```
1 import cat
2
3 c=mymodule.cat["Name"]
4 print(c)
5 |
```

## The import statement

TLDR:

```
1 # Bad
2 [...]
3 from mymodule import *
4 [...]
5 x=pow(4,2)# is pow part of mymodule? # A
  builtin? Defined above?
6
7 #Better
8 [...]
9 from mymodule import pow
10 x=pow(4,2)# pow maybe part of
  mymodule,if not redefined above
11
12 #Best
13 [...]
14 import mymodule
15 x=mymodule.pow(4,2) #pow is visibly
  part of mymodule namespace
```

Python import statement allows you to import a module into your code. They are similar to `#include` header files in C/C++.

When the python interpreter comes across an import statement, it imports the module if it finds the module in the search path.

Here search path means the list of directories that the interpreter searches through before importing the modules.

```
>>> sys.path
['', 'C:\\Users\\Vanshi\\AppData\\Local\\Programs\\Python\\Python38-32\\Lib\\idlelib', 'C:\\Users\\Vanshi\\AppData\\Local\\Programs\\Python\\Python38-32\\python38.zip', 'C:\\Users\\Vanshi\\AppData\\Local\\Programs\\Python\\Python38-32\\DLLs', 'C:\\Users\\Vanshi\\AppData\\Local\\Programs\\Python\\Python38-32\\lib', 'C:\\Users\\Vanshi\\AppData\\Local\\Programs\\Python\\Python38-32', 'C:\\Users\\Vanshi\\AppData\\Roaming\\Python\\Python38\\site-packages', 'C:\\Users\\Vanshi\\AppData\\Local\\Programs\\Python\\Python38-32\\lib\\site-packages', 'C:\\Users\\Vanshi\\AppData\\Local\\Programs\\Python\\Python38-32\\lib\\site-packages\\win32', 'C:\\Users\\Vanshi\\AppData\\Local\\Programs\\Python\\Python38-32\\lib\\site-packages\\win32\\lib', 'C:\\Users\\Vanshi\\AppData\\Local\\Programs\\Python\\Python38-32\\lib\\site-packages\\Pythonwin']
```

The above image shows the list of directories that the interpreter went through before importing the module.

## How to import a module

Following are the ways to import a module:

To import the entire module:

Use statement `import module name`

Example



```
import math
print(math.pow(4,2))
print(math.log10(10))
```

16.0  
1.0  
> |

The import math statement will make all the functions under the math module available to the program

Here we are accessing the module “math”

The functions pow() and log() are defined in the math module. Dot operator is used here

To import the entire module you can also use

Use statement from module\_name import \*

from module\_name import\* way is generally not used by the coders as it makes it harder to track down a problem. Because we won't be able to know whether the function is imported and where it came from.

```
1  # importing sqrt() and factorial from
   the
2  # module math
3  from math import *
4
5  # if we simply do "import math", then
6  # math.sqrt() and math.factorial()
7  # are required.
8  print(sqrt(25))
9  print(factorial(5))
10
```

Here, no dot operator has been used as done in the former. We only wrote function names before arguments.

To import specific functions/variables/classes

Use statement `from module_name import function_name/class_name/variables_name`

```
1  # importing pow() and factorial from the
2  # module math
3  from math import pow, factorial
4
5  # if we simply do "import math", then
6  # math.pow(4,2) and math.factorial()
7  # are required.
8  print(pow(4,2))
9  print(factorial(6))
10
```

Here also we didn't use the dot operator. And simply called the mathematical functions with arguments.

## Python module search path

By default python interpreter searches through the following directories when a module is imported:

1. Current directory
2. Environment variable path
3. Directories installed from 3rd parties

The interpreter will throw an error if the imported module is not found in any of the above directories.

If you do not want to put your module file in the current directory then you can find the path by going to advanced system <setting><environment variables><path>. You can add your module file in that path also

To find the module path write in your code

`sys(module_name)`

## Naming a module

User should name their modules while adhering to the following suggestions:

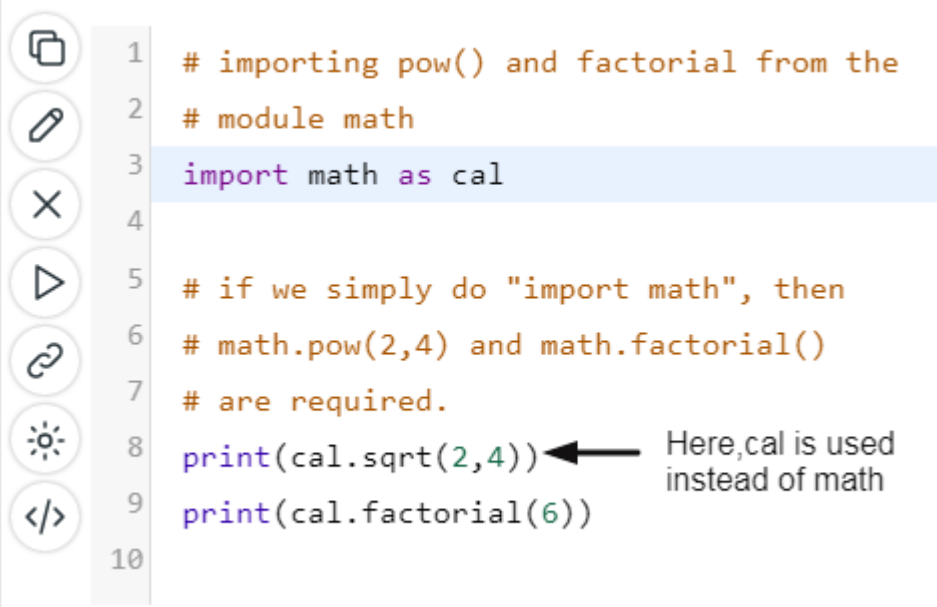
1. It should be short and, lowercased.
2. Avoid using special symbols like ( \_,?,., etc). If you name your module “my.module.py” the python expects to find the module.py file in the folder named “my” which is not the case.

## Re-naming a module

If the module name is too long then we can use the keyword as to rename the module.

Syntax :`import module_initial name as module_updated name`

Example



```
1  # importing pow() and factorial from the
2  # module math
3  import math as cal
4
5  # if we simply do "import math", then
6  # math.pow(2,4) and math.factorial()
7  # are required.
8  print(cal.sqrt(2,4))
9  print(cal.factorial(6))
10
```

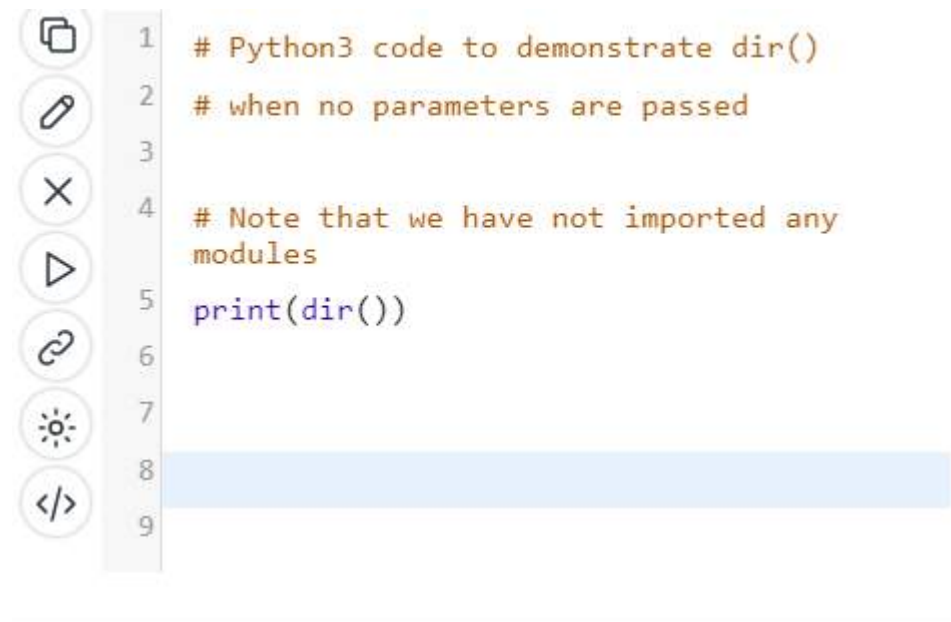
Here, cal is used instead of math

## The dir() built in function

*dir([object])*

The dir() function returns attributes & values in local scope.

If it is used without any arguments, then it returns all the attributes and values in the local scope.

A screenshot of a Python IDE interface. On the left, there is a vertical toolbar with icons for file operations (copy, paste, delete), editing (pencil), execution (play), and settings (gear). The main area shows a code editor with the following Python code:

```
1  # Python3 code to demonstrate dir()  
2  # when no parameters are passed  
3  
4  # Note that we have not imported any  
   modules  
5  print(dir())  
6  
7  
8  
9
```

Line 8 is highlighted with a light blue background.

**Output :**

A screenshot of the output of the Python code. It shows a list of strings representing the attributes and values in the local scope:

```
['__builtins__', '__cached__', '__doc__', '__file__',  
                                '__name__']
```

If a module is passed in the arguments then it will return everything that exists inside that module.

Example

```

# Python3 code to demonstrate dir() function
# when a module Object is passed as parameter.

# import the random module
import random

# Prints list which contains names of
# attributes in random function
print("The contents of the random library are::")

# module Object is passed as parameter
print(dir(random))

```

### Output :

The contents of the random library are ::

```

['BPF', 'LOG4', 'NV_MAGICCONST', 'RECIP_BPF', 'Random',
'SystemRandom', 'TWOPI', '_BuiltinMethodType', '_Metl',
'_Set', '__all__', '__builtins__', '__cached__', '__c',
'__name__', '__package__', '__spec__', '_acos', '_ce',
'_inst', '_log', '_pi', '_random', '_sha512', '_sin',
'_urandom', '_warn', 'betavariate', 'choice', 'expov',
'getrandbits', 'getstate', 'lognormvariate', 'normal',
'random', 'randrange', 'sample', 'seed', 'setstate',
'vonmisesvariate', 'weibullvariate']

```

To see what attributes exist in a user-defined class.

Create a class and then attributes in it

Write `dir(class_name)` to see the list of attributes present in the class.

```
1 # Python3 program to demonstrate working
2 # of dir(), when user defined objects are
3 # passed are parameters.
4
5
6 # Creation of a simple class with __dir__()
7 # method to demonstrate it's working
8 class you:
9     talent=100
10    skills=100
11    print(dir(you))
12
```

Output:

Copy

```
['_setattr_', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'skills', 'talent']
```

User defined attributes. →

## Summary

